REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
10-04-2015	Final Report	15-Oct-2009 - 14-Oct-2013

4. TITLE AND SUBTITLE
Final Report: Compiler-Driven Performance Optimization and Tuning for Multicore Architectures

5a. CONTRACT NUMBER
W911NF-10-1-0004

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
106011

6. AUTHORS
Jagannathan Ramanujam

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
Louisiana State University and A&M Colleg
Office of Sponsored Programs
Louisiana State University and A&M College
Baton Rouge, LA 70803 -0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
56889-CS-DPS.19

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT
The widespread emergence of multicore processors as the computing engine in all commodity platforms presents our field with an enormous software development crisis. For over two decades, sequential software applications have enjoyed the free-ride of performance improvement with each new processor generation. The reality today is that existing and new applications must be changed to make them multi-threaded if they are to experience any performance benefits from newer generations of processors. An attractive approach to addressing some aspects of this monumental software development challenge is to develop highly optimized

15. SUBJECT TERMS
multicore; compiler optimizations; tiling; GPUs

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jagannathan Ramanujam
UU	UU	UU	UU		19b. TELEPHONE NUMBER 225-578-5628

Report Title

Final Report: Compiler-Driven Performance Optimization and Tuning for Multicore Architectures

ABSTRACT

The widespread emergence of multicore processors as the computing engine in all commodity platforms presents our field with an enormous software development crisis. For over two decades, sequential software applications have enjoyed the free-ride of performance improvement with each new processor generation. The reality today is that existing and new applications must be changed to make them multi-threaded if they are to experience any performance benefits from newer generations of processors. An attractive approach to addressing some aspects of this monumental software development challenge is to develop highly optimized version of compute-intensive portions of an application. This project aims to develop a powerful system for auto-tuning of library routines and compute-intensive kernels, driven by the Pluto system for multicores that we are developing. The work here is motivated by recent advances in two major areas of performance optimization: (i) auto-tuning approaches that employ a combination of static and dynamic exploration of the optimization space, and (ii) polyhedral-based approaches for powerful transformations of affine computations.

Our work builds on some very recent developments using polyhedral models showing great promise for developing effective automatic parallelization frameworks for multicore architectures. With the polyhedral model, it is possible to reason about the correctness of complex loop transformations in a completely mathematical setting using powerful machinery from linear algebra and linear programming. This enables effective integrated transformation, and therefore can be the basis for developing a very powerful automatic parallelization framework that can target different multicore platforms. The project addresses a number of key issues that are very important in developing an automatic parallelization and data locality optimization framework that is effective over a range of user application codes: model-driven search for determination of effective tile sizes and loop fusion choices; exploration of dynamic scheduling of tiles; extended tiling approaches like overlapped/split tiles to enhance concurrency; automatic generation of parallel code for accelerators with multiple distinct address spaces; and development of an extensive benchmark suite for assessment of automatic parallelization systems. This novel optimization framework has potential for high payoffs in generating high-performance code.

Enter List of papers submitted or published that acknowledge ARO support from the start of the project to the date of this printing. List the papers, including journal references, in the following categories:

(a) Papers published in peer-reviewed journals (N/A for none)

Received		Paper
04/10/2015	14.00	Hassan Salamy, J. Ramanujam. Storage Optimization through Offset Assignment with Variable Coalescing, ACM Transactions on Embedded Computing Systems, (06 2012): 0. doi: 10.1145/2180887.2180893
04/10/2015	15.00	S. Tavarageri, J. Ramanujam, P. Sadayappan. Adaptive parallel tiled code generation and accelerated auto-tuning, International Journal of High Performance Computing Applications, (07 2013): 0. doi: 10.1177 /1094342013493939
09/28/2012	7.00	Hassan Salamy, J. Ramanujam. An ILP solution to address code generation for embedded applications on digital signal processors, ACM Transactions on Design Automation of Electronic Systems, (06 2012): 0. doi: 10.1145/2209291. 2209301
09/28/2012	8.00	Hassan Salamy, J. Ramanujam. An Effective Solution to Task Scheduling and Memory Partitioning for Multiprocessor System-on-Chip, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, (05 2012): 0. doi: 10.1109/TCAD.2011.2181848
09/28/2012	9.00	Qingda Lu, Xiaoyang Gao, Sriram Krishnamoorthy, Gerald Baumgartner, J. Ramanujam, P. Sadayappan. Empirical performance model-driven data layout optimization and library call selection for tensor contraction expressions, Journal of Parallel and Distributed Computing, (03 2012): 0. doi: 10.1016/j.jpdc.2011.09.006
09/28/2012	12.00	Hassan Salamy, J. Ramanujam. CODE SIZE REDUCTION FOR ARRAY INTENSIVEAPPLICATIONS ON DIGITAL SIGNAL PROCESSORS, JOURNAL OF Circuits, Systems and Computers, (05 2012): 1. doi:
TOTAL:		**6**

Number of Papers published in peer-reviewed journals:

(b) Papers published in non-peer-reviewed journals (N/A for none)

Received		Paper
TOTAL:		

Number of Papers published in non peer-reviewed journals:

(c) Presentations

T. Henretty, R. Veras, F. Franchetti, L.N. Pouchet, J. Ramanujam and P. Sadayappan, "A Domain-Specific Language and Compiler for Stencil Computations on Short-Vector SIMD and GPU Architectures," in 17th Workshop on Compilers for Parallel Computing (CPC 2013), Lyon, France, July 2013.

A. Panyala, P. Bhattacharya, G. Baumgartner, and J. Ramanujam, "A fusion-based optimization framework for a tensor contraction language," in Workshop on Parallel Matrix Algorithms and Applications (PMAA 2012), London, UK, June 2012.

S. Tavarageri, L.-N. Pouchet, J. Ramanujam, A. Rountev, J. Ramanujam, and P. Sadayappan, "Dynamic selection of tile sizes," in 16th International Workshop on Compilers for Parallel Computing (CPC 2012), Padova, Italy, January 2012.

J. Ramanujam, "EvolveTile: Dynamic Selection of Tile Sizes," Workshop on Libraries and Automatic Tuning for Extreme Scale Systems, Lake Tahoe, CA. August 2011.

J. Ramanujam, "The Tensor Contraction Engine (TCE): A Domain-Specific Approach to Synthesizing High-Performance Codes for Quantum Chemistry," Imperial College, London, UK, June 2011.

J. Ramanujam, "High-Level Programming and Compilation Technology for GPUs," Ecole Normale Superiure de Lyon, France, June 2011.

S. Tavarageri, A. Hartono, M. Baskaran, L.-N. Pouchet, J. Ramanujam, and P. Sadayappan, "Parametric Tiling of Affine Loop Nests," in 15th Workshop on Compilers for Parallel Computers (CPC 2010), Vienna, Austria, July 2010.

M. Baskaran, A. Hartono, L.-N. Pouchet, S. Tavarageri, J. Ramanujam, and P. Sadayappan, "Parametric Tiling for Autotuning," in Workshop on Parallel Matrix Algorithms and Applications (PMAA 2010), Basel, Switzerland, July 2010.

J. Ramanujam, "Optimizations for Multicore and GPGPU Architectures," at the Spring School on Beyond the PC: Application-Specific Systems: Design and Implementation (six hours of lecture), Lyon, France, February 15–19, 2010.

Number of Presentations: 9.00

Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Received	Paper
04/10/2015	13.00 S. Tavarageri, A. Hartono, M. Baskaran, L.-N. Pouchet, J. Ramanujam, P. Sadayappan. Parametric Tiling of Affine Loop Nests, 15th Workshop on Compilers for Parallel Computers (CPC 2010). 07-JUL-10, . : ,

TOTAL: **1**

Number of Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Peer-Reviewed Conference Proceeding publications (other than abstracts):

Received	Paper	
01/09/2012	1.00	Kevin Stock, , Tom Henretty, , Louis-Noël Pouchet, , Franz Franchetti, , J. Ramanujam,, P. Sadayappan. Data Layout Transformation for Stencil Computations on Short-Vector SIMD Architectures , Compiler Construction - 20th International Conference, CC 2011. 30-MAR-11, . : ,
01/09/2012	2.00	Uday Bondhugula, Cédric Bastoul, Albert Cohen, J. Ramanujam, P. Sadayappan, Nicolas Vasilache, Louis-Noël Pouchet. Loop Transformations: Convexity, Pruning and Optimization, 38th annual ACM SIGPLAN-SIGACT symposium on Programming Languages. 25-JAN-11, Austin, Texas, USA. : ,
01/09/2012	3.00	Louis-Noel Pouchet, Uday Bondhugula, Cedric Bastoul, Albert Cohen, J. Ramanujam, P. Sadayappan. Combined Iterative and Model-driven Optimization in an Automatic Parallelization Framework, 2010 SC - International Conference for High Performance Computing, Networking, Storage and Analysis. 12-NOV-10, New Orleans, LA, USA. : ,
01/09/2012	4.00	Muthu Manikandan Baskaran, Albert Hartono, Sanket Tavarageri, Thomas Henretty, J. Ramanujam, P. Sadayappan. Parameterized tiling revisited, 8th annual IEEE/ ACM international symposium on Code Generation and Optimization. 23-APR-10, Toronto, Ontario, Canada. : ,
01/10/2012	5.00	Albert Hartono, Muthu Manikandan Baskaran, J. Ramanujam, P. Sadayappan. DynTile: Parametric tiled loop generation for parallel execution on multicore processors, 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS). 18-APR-10, Atlanta, GA, USA. : ,
01/10/2012	6.00	Tong-Chai Wang, J. Ramanujam. A dynamic heuristic algorithm for offset assignment, 2010 International Computer Symposium (ICS 2010). 15-DEC-10, Tainan, Taiwan. : ,
04/10/2015	16.00	John Eisenlohr, Louis-Noel Pouchet, Mahesh Ravishankar, J. Ramanujam, Atanas Rountev, P. Sadayappan. Code generation for parallel execution of a class of irregular loops on distributed memory systems, 2012 SC - International Conference for High Performance Computing, Networking, Storage and Analysis. 09-NOV-12, Salt Lake City, UT, USA. : ,
04/10/2015	18.00	Tobias Grosser, Albert Cohen, Paul H. J. Kelly, J. Ramanujam, P. Sadayappan, Sven Verdoolaege. Split tiling for GPUs, the 6th Workshop. 15-MAR-13, Houston, Texas. : ,
04/10/2015	17.00	Tom Henretty, Richard Veras, Franz Franchetti, Louis-Noël Pouchet, J. Ramanujam, P. Sadayappan. A stencil compiler for short-vector SIMD architectures, the 27th international ACM conference. 09-JUN-13, Eugene, Oregon, USA. : ,
09/28/2012	10.00	Sanket Tavarageri, Louis-Noel Pouchet, J. Ramanujam, Atanas Rountev, P. Sadayappan. Dynamic selection of tile sizes, 2011 18th International Conference on High Performance Computing (HiPC). 17-DEC-11, Bengaluru, India. : ,
09/28/2012	11.00	Jun Shirako, Kamal Sharma, Naznin Fauzia, Louis-Noel Pouchet, J. Ramanujam, P. Sadayappan, Vivek Sarkar. Analytical Bounds for Optimal Tile Size Selection, Compiler Construction - 21st International Conference, CC 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings. 24-MAR-12, . : ,

TOTAL: 11

Number of Peer-Reviewed Conference Proceeding publications (other than abstracts):

(d) Manuscripts

Received Paper

TOTAL:

Number of Manuscripts:

Books

Received Book

TOTAL:

Received Book Chapter

TOTAL:

Patents Submitted

Patents Awarded

Awards

Graduate Students

NAME	PERCENT_SUPPORTED	Discipline
Santosh Verma	0.50	
Mohammad Rastegar Tohid	0.50	
Sahar Marefat Navaz	0.50	
FTE Equivalent:	**1.50**	
Total Number:	**3**	

Names of Post Doctorates

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Faculty Supported

NAME	PERCENT_SUPPORTED	National Academy Member
Jagannathan Ramanujam	0.22	
FTE Equivalent:	**0.22**	
Total Number:	**1**	

Names of Under Graduate students supported

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Student Metrics

This section only applies to graduating undergraduates supported by this agreement in this reporting period

The number of undergraduates funded by this agreement who graduated during this period: 0.00

The number of undergraduates funded by this agreement who graduated during this period with a degree in science, mathematics, engineering, or technology fields:..... 0.00

The number of undergraduates funded by your agreement who graduated during this period and will continue to pursue a graduate or Ph.D. degree in science, mathematics, engineering, or technology fields:..... 0.00

Number of graduating undergraduates who achieved a 3.5 GPA to 4.0 (4.0 max scale):..... 0.00

Number of graduating undergraduates funded by a DoD funded Center of Excellence grant for Education, Research and Engineering:..... 0.00

The number of undergraduates funded by your agreement who graduated during this period and intend to work for the Department of Defense 0.00

The number of undergraduates funded by your agreement who graduated during this period and will receive scholarships or fellowships for further studies in science, mathematics, engineering or technology fields:..... 0.00

Names of Personnel receiving masters degrees

NAME
Total Number:

Names of personnel receiving PHDs

NAME	
Santosh Verma	
Total Number:	**1**

Names of other research staff

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Sub Contractors (DD882)

Inventions (DD882)

Scientific Progress

Technology Transfer

We have had interactions with scientists from Pacific Northwest National Labs.

Compiler-Driven Performance Optimization and Tuning for Multicore Architectures
PI: J. Ramanujam (Louisiana State University)
Proposal Number: 56889-CS-DPS, Agreement Number: W911NF-10-1-0004
Final Report – Scientific Progress and Accomplishments: October 15, 2009–October 14, 2013

This portion the report briefly describes the major aspects of scientific progress and accomplishments during this project spanning four years.

The widespread emergence of multicore processors as the computing engine in all commodity platforms presents our field with an enormous software development crisis. For over two decades, sequential software applications have enjoyed the free-ride of performance improvement with each new processor generation. The reality today is that existing and new applications must be changed to make them multi-threaded if they are to experience any performance benefits from newer generations of processors. An attractive approach to addressing some aspects of this monumental software development challenge is to develop highly optimized version of compute-intensive portions of an application. This project aims to develop a powerful system for auto-tuning of library routines and compute-intensive kernels, driven by the Pluto system for multicores that we are developing. The work here is motivated by recent advances in two major areas of performance optimization: (i) auto-tuning approaches that employ a combination of static and dynamic exploration of the optimization space, and (ii) polyhedral-based approaches for powerful transformations of affine computations.

Our work builds on some very recent developments using polyhedral models showing great promise for developing effective automatic parallelization frameworks for multicore architectures. With the polyhedral model, it is possible to reason about the correctness of complex loop transformations in a completely mathematical setting using powerful machinery from linear algebra and linear programming. This enables effective integrated transformation, and therefore can be the basis for developing a very powerful automatic parallelization framework that can target different multicore platforms. The project addresses a number of key issues that are very important in developing an automatic parallelization and data locality optimization framework that is effective over a range of user application codes: model-driven search for determination of effective tile sizes and loop fusion choices; exploration of dynamic scheduling of tiles; extended tiling approaches like overlapped/split tiles to enhance concurrency; automatic generation of parallel code for accelerators with multiple distinct address spaces; and development of an extensive benchmark suite for assessment of automatic parallelization systems. This novel optimization framework has potential for high payoffs in generating high-performance code.

Next, this report describes the activities that we engaged in during this project.

1. Performance Optimizations for Multicore GPUs

Graphics Processing Units (GPUs) offer tremendous computational power. CUDA (Compute Unified Device Architecture) provides a multi-threaded parallel programming model, facilitating high performance implementations of general-purpose computations. However, the explicitly managed memory hierarchy and multi-level parallel view make manual development of high-performance CUDA code rather complicated. Hence the automatic transformation of sequential input programs into efficient parallel CUDA programs is of considerable interest.

Recently, we [BRS 10] have developed an end-to-end automatic C-to-CUDA code generator using a polyhedral compiler transformation framework. We have used and adapted PLUTO (our state-of-the-art tool for polyhedral compilation) and other publicly available tools that have made polyhedral compiler optimization practically effective. The result is a C-to-CUDA transformation system that generates two-level parallel CUDA code that is optimized for efficient data access. We evaluated the quality of the generated code using several benchmarks, by comparing the performance of automatically generated CUDA code with hand-tuned CUDA code where available and also with optimized code generated by the Intel ICC compiler for a general-purpose multicore CPU. Experimental results demonstrated the performance improvements achieved using the framework.

2. Compiler-Assisted Dynamic Scheduling for Effective Parallelization of Tiled Loop Nests

For multi-statement input programs with statements of different dimensionalities, such as Cholesky or LU decomposition, the parallel tiled code generated by PLUTO may suffer from significant load imbalance as well excessive barrier synchronization overheads, resulting in poor scalability on multi-core systems. Recently, we have developed a completely automatic parallelization approach for transforming input affine sequential codes into efficient parallel codes that can be executed on a multi-core system in a load-balanced manner. In our approach, we employ a compile-time technique that enables dynamic extraction of inter-tile dependences at run-time, and dynamic scheduling of the parallel tiles on the processor cores for improved scalable execution. Our approach obviates the need for programmer intervention and re-writing of existing algorithms for efficient parallel execution on multi-cores. We have demonstrated the usefulness of our approach through comparisons using linear algebra computations: LU and Cholesky decomposition. In addition, we have used this dynamic scheduling approach with *DynTile,* a parallel parametric tiling strategy that we have developed.

3. Model-driven Optimization

Tiling is a critical transformation for performance optimization, and a number of recent efforts have focused on advanced techniques for generating parametrically tiled code. However, the problem of selecting effective tile sizes remains challenging. The current state of practice, as exemplified by systems such as ATLAS, is expensive empirical search. Several previous studies have also proposed analytical models, but none have been shown to be practically effective in selecting the best tile sizes.

Recently, in collaboration with Vivek Sarkar (one of the co-PIs on the DARPA-funded PACE project) and Jun Shirako from Rice University and P. Sadayappan from Ohio State, we are in th eprocess of developing a novel approach to model-driven empirical search for tile size selection. Two analytical models have been used, one a conservative model based on the cache footprint of a tile that does not account for reuse, and the other, an aggressive model that assumes maximal intra-tile reuse. Empirical search is carried out in the search space bounded between regions delineated by the two models. Experimental results on multiple platforms have demonstrated the practical effectiveness of the new approach to model-driven empirical search for tile sizes.

4. Combined Iterative and Model-Driven Optimization

The multi-core era places significant demands on an optimizing compiler, which must parallelize programs, exploit memory hierarchy, and leverage the ever-increasing SIMD capabilities of modern processors. Existing model-based heuristics for performance optimization used in compilers are limited in their ability to identify profitable parallelism/locality trade-offs and usually lead to sub-optimal performance. To address this problem, in [PB+ 10], we have developed a completely automatic framework in which we focus the empirical search on the set of valid possibilities to perform fusion/code motion, and rely on model-based mechanisms to perform tiling, vectorization and parallelization on the transformed program. This paper addressed the problem of optimizing and parallelizing programs automatically, focusing on static control loop nests. Our approach departs from the traditional best-effort compiler optimizations, aiming for performance portability across a variety of shared-memory multiprocessors. We proposed a combined iterative and model-driven approach, leveraging a state-of-the-art parallelization method based on loop tiling, and combining it with a novel feedback-directed scheme for loop fusion and distribution.

Our technique builds an expressive search space of loop transformation sequences, expressed in the polyhedral model as a set of affine scheduling functions. The search space encompasses complex compositions of loop transformations, including loop fusion and distribution, loop tiling for parallelism and locality (caches, registers), loop interchange, and loop shifting (pipelining). We proposed a convex encoding of all legal transformed program versions as the space to search.

We demonstrated the effectiveness of this approach in terms of strong performance improvements on a single target as well as performance portability across different target architectures. We performed experiments on three different platforms: a 24-core Xeon, a 16-core Opteron, and a single-core low-power Atom processor. Our experiments confirm that no single program version performs equally well on different targets, with penalties reaching $2\times$ when running the

best version for a given target on a different target. We also consistently demonstrate strong performance improvements over the state-of-the-art model-based compilers, with performance improvement factors up to 8× over Intel's compiler.

5. Characterization of Affine Transformations

High-level loop transformations are a key instrument in mapping computational kernels to effectively exploit resources in modern processor architectures. However, determining appropriate compositions of loop transformations to achieve this remains a significantly challenging task; current compilers may achieve significantly lower performance than hand-optimized programs.

Despite four decades of research and development in compiler optimizations, it remains a challenging task for compiler designers and a frustrating experience for programmers — the performance gap, between expert-written code for a given machine and that optimized and parallelized automatically by a compiler, is *widening* with newer generations of hardware. One reason for this widening gap is due to the way loop nest optimizers attempt to decompose the global optimization problem into simpler sub-problems. The polyhedral compiler framework provides a convenient and powerful abstraction to apply composite transformations in one step. However, the space of affine transformations is extremely large. Linear optimization in such a high-dimensional space raises complexity issues and its effectiveness is limited by the lack of accurate profitability models.

In [PB+ 11], we made the following contributions:

- We provided a *complete, convex formalization* for affine transformations. This convex set of semantics-preserving, distinct, affine multidimensional schedules opens the door to more powerful linear optimization algorithms.
- We proposed a decomposition of the general optimization problem over this convex polyhedron into sub-problems of much lower complexity, introducing the powerful *fusibility* concept. The fusibility relation generalizes the legality conditions for loop fusion, abstracting a large set of multidimensional, affine, fusion-enabling transformations.
- We demonstrated that exploring fusibility opportunities in a loop nest reduces to the enumeration of total preorders, and to the existence of pairwise compatible loop permutations. We also studied the transitivity of the fusibility relation.
- Based on these results, we designed a multidimensional affine scheduling algorithm targeted at achieving portable high performance across modern processor architectures.

Our approach has been fully implemented in a source-to-source compiler and validated on representative benchmarks.

6. Data Layout Transformations for Stencil Computations on Short-vector SIMD Architectures

Short vector SIMD extensions are included in all major high-performance CPUs. While ubiquitous, the ISA and performance characteristics vary from vendor to vendor and across hardware generations. For instance, Intel has introduced SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, and LRBni ISA extensions over the years. With every processor, the latency and throughput numbers of instructions in these extensions change. IBM, Freescale, and Motorola have introduced AltiVec, VMX, VMX128, VSX, Cell SPU, PowerXCell 8i SPU SIMD implementations. In some instances (RoadRunner, BlueGene/L), custom ISA extensions were designed since the supercomputing installation was big enough to warrant such an investment. These extensions provide from 2-way adds and multiplies up to 16-way fused multiply-add operations, promising significant speed-up. It is therefore important to optimize for these extensions.

Stencil computations represent an important class, occuring in many image processing applications, computational electromagnetics and solution of PDEs using finite difference or finite volume discretizations. There has been considerable recent interest in optimization of stencil computations. In [HS+ 11], we focus on the problem of optimizing single-core performance on modern processors with SIMD instruction sets. Stencil computations are readily expressible in a form with vectorizable innermost loops where arrays are accessed at unit stride. However, as elaborated upon

later, there is a fundamental performance limiting factor with all current short-vector SIMD instruction sets such as SSE, AVX, VMX, etc. We formalize the problem through the abstraction of stream alignment conflicts. We note that the alignment conflict issue we formulate and solve in this paper pertains to algorithmic alignment constraints and is distinctly different from the previously studied topic of efficient code generation on SIMD architectures with hardware alignment constraints. We address the problem of resolving stream alignment conflict through the novel use of a nonlinear data layout transformation and develop a compiler framework to identify and suitably transform the computations. Experimental results on multiple targets demonstrate the effectiveness of the approach on a number of stencil kernels.

7. Parametric Tiling

Tiling is a crucial loop transformation for generating high performance code on modern architectures. Efficient generation of multi-level tiled code is essential for maximizing data reuse in systems with deep memory hierarchies. A number of works on tiling generate tiled code where the tile sizes are fixed. Since the performance of tiled code varies greatly with the choice of tile sizes, it is preferable to specify the tile sizes as runtime parameters in the code. In such cases, parametric tiling refers to the generation of tiled code where tile loops are runtime parameters. For example, such an approach would enable empirical search for tile sizes. The importance of parametric tiling is exemplified by the highly successful ATLAS system for empirical tuning of BLAS kernels. But the ATLAS system can only tune BLAS kernels. Further, it was manually engineered by experts with insights into tiling for optimization of BLAS kernels. There has been much recent interest in developing generalized tuning systems that can similarly tune and optimize codes input by users or library developers. An efficient parametric tiling tool is extremely valuable for generating input tiled codes for such empirical tuning systems.

Tiled loops with parametric tile sizes (not compile-time constants) facilitate runtime feedback and dynamic optimizations used in iterative compilation and automatic tuning. Previous parametric multi-level tiling approaches have been restricted to perfectly nested loops, where all assignment statements are contained inside the innermost loop of a loop nest. Previous solutions to tiling for imperfect loop nests have only handled fixed tile sizes. Last year, we [HB+ 09] presented an approach to parametric multi-level tiling of imperfectly nested loops. The tiling technique generates loops that iterate over full rectangular tiles, making them amenable to compiler optimizations such as register tiling.

The solution presented in [HB+ 09] (called PrimeTile) works for sequential tiling. We have developed two parallel parametric tiling techniques that handle imperfectly nested loops: a dynamic solution [HB+ 10] (called DynTile) and a compile-time solution (called PTile) that also included the development of a relaxed symbolic Fourier-Motzkin elimination technique [BH+ 10].

In [TH+ 10], we present a comparative study of three recently developed approaches for parametric tiling of affine loop nests:

- **PrimeTile** [HB+ 09]: This was the first system to generate parametrically tiled code for affine imperfectly nested loops. As explained in the next section, it uses a level by level approach to generate tiled code, with a prolog, epilog, and a full-tiles loop nest corresponding to each nesting level of the original code. The approach was limited to sequential output tiled code.

- **DynTile** [HB+10] : This was the first implementation of an approach for parallel execution of parametrically tiled affine code. It utilizes wavefront parallelism in the tiled iteration space corresponding to the convex hull of all the statement domains of the input untiled code. Wavefront parallelism in the tiled iteration space is exploited through use of an inspector/executor approach. A statically generated inspector code scans the tiles and places them in bins corresponding to the different wavefronts. Dynamic scheduling of the tiles is then performed in order of increasing wavefronts.

- **PTile** [BH+ 10]: This was the first approach to compile-time generation of code for wavefront-parallel tiled execution. Instead of the dynamic runtime scanning and binning of tiles as done by the DynTile approaches, transformed code for wavefront parallel tiled execution is automatically generated.

We observed that for sequential execution, PrimeTile performs best in our experiments, with a performance often comparable with DynTile, in particular when considering ICC. We also observe that GCC has more trouble in optimizing the code generated by DynTile than the one generated by PrimeTile. The slowdown of PTile is explained by the

order in which tiles are executed: PTile generates a wavefront outer-most tile loop, that is needed to ensure correctness of parallel tile execution. However, for sequential execution, this wavefront order is detrimental to locality, and thus to performance.

Also, we observed that DynTile slightly outperforms PTile for parallel execution. The explanation lies in the amount of parallelism that is exploited by the two approaches. For DynTile, all tiles that can be executed in parallel for a given wavefront are indeed marked as parallel, with a `#pragma omp for` around the loop which iterates over all tiles belonging to the wavefront. However, for PTile, only the next outer-most loop is marked as parallel, the outer-most loop being the wavefront loop. So even though the remaining tile loops are also parallel, this parallelism is not exploited. It is expected that coalescing the parallel tile loops into a single loop would improve load balancing, and exploit all the available parallelism.

In addition, we observed that for all considered benchmarks, neither ICC nor GCC was able to vectorize loops in the computational kernel. For most cases, the compiler was unable to analyze the loops because of complex controls and/or access patterns. The increased complexity of controls generated by polyhedral parametric tiling challenges the compiler's loop analyzer, and it has been shown that for such cases it is beneficial to expose and explicitly mark vectorizable loops from within the polyhedral transformation framework using such as Pluto.

8. Adaptive Parametric Tiling

Tiling is a crucial loop transformation for generating high performance code on modern architectures. Efficient generation of multi-level tiled code is essential for maximizing data reuse in systems with deep memory hierarchies. A number of works on tiling generate tiled code where the tile sizes are fixed. Since the performance of tiled code varies greatly with the choice of tile sizes, it is preferable to specify the tile sizes as runtime parameters in the code. In such cases, parametric tiling refers to the generation of tiled code where tile loops are runtime parameters. For example, such an approach would enable empirical search for tile sizes. The importance of parametric tiling is exemplified by the highly successful ATLAS system for empirical tuning of BLAS kernels. But the ATLAS system can only tune BLAS kernels. Further, it was manually engineered by experts with insights into tiling for optimization of BLAS kernels. There has been much recent interest in developing generalized tuning systems that can similarly tune and optimize codes input by users or library developers. An efficient parametric tiling tool is extremely valuable for generating input tiled codes for such empirical tuning systems.

Tiled loops with parametric tile sizes (not compile-time constants) facilitate runtime feedback and dynamic optimizations used in iterative compilation and automatic tuning. Previous parametric multi-level tiling approaches have been restricted to perfectly nested loops, where all assignment statements are contained inside the innermost loop of a loop nest. Previous solutions to tiling for imperfect loop nests have only handled fixed tile sizes. Earlier in this project, we [HB+ 09] presented an approach to sequential parametric multi-level tiling of imperfectly nested loops. The tiling technique generates loops that iterate over full rectangular tiles, making them amenable to compiler optimizations such as register tiling. Later, we developed two parallel parametric tiling techniques that handle imperfectly nested loops: a dynamic solution [HB+ 10] (called DynTile) and a compile-time solution (called PTile) that also included the development of a relaxed symbolic Fourier-Motzkin elimination technique. In [TH+ 10], we presented a comparative study of three recently developed approaches for parametric tiling of affine loop nests.

Advances in parameterized tiling, where tile sizes are symbolic parameters have enabled the generation of code that can be executed with different tile sizes in different environments. There has also been much interest in developing analytical models to determine optimal tile sizes as a function of architectural parameters (sizes and associativities of caches and TLBs, etc.) and code characteristics. However, so far no analytical model has been shown to be effective in finding optimal tile sizes across a wide range of programs and target platforms. Therefore, the current state-of-practice for effective tile size selection is to use auto-tuning, where an empirical search for the best tile sizes (often off-line) is carried out by running the program with different tile sizes, and the best performing configuration observed is retained. While ATLAS is perhaps the most widely known auto-tuning framework, many other systems have demonstrated the effective use of auto-tuning for selection of tile sizes and other optimization parameters such as the degree of loop unrolling.

Although auto-tuning is effective for tile size optimization, it requires execution of a large number of tuning runs on the target platform prior to the actual "production" run of an application, and thus is more suited for the development

of library packages, rather than for user code. Further, auto-tuning may not be feasible in contexts such as cloud computing, where users run their applications in a cost-effective manner in the "cloud," but the characteristics of the machine(s) on which the program is executed may not be known to the user beforehand. Depending on the number of other applications running on the same machine (a factor that is often beyond the control of application users), every run of the application may require different tile sizes for best performance. Furthermore, different runs of an application could be on different platforms. Autotuning based on empirical search of the space of possible tile sizes is thus infeasible in this context.

Tiling is an important program transformation that is often used to enhance cache locality and to obtain coarse-grained parallelism. Recently [TP+ 11], [TR+ 13], we have developed a novel approach for the generation of tiled code, so that different tile sizes are dynamically tried out as the actual "production" code (that is parametrically tiled) executes, and an effective set of tile sizes is determined on the fly. In other words, there is no need to rely on a priori empirical search and auto-tuning. Instead, the program observes and adapts its own run-time behavior, as a function of the (unknown) machine characteristics and system load. We developed a new code generation techniques to enable on-the-fly variation of the tile sizes during program execution. The developed approach can handle affine multi-statement imperfectly nested loop nests. In addition, we presented an algorithm for dynamic adaptation of tile sizes. Our approach works by moving towards better tile sizes by monitoring the performance of a few iterations of the loop. We found that a run-time adaptation of tile sizes is able to achieve near-optimal performance for a number of benchmarks.

9. Layout Optimization and Library Call Selection in Code Generation

The Tensor Contraction Engine (TCE) is a compiler that translates high-level, mathematical tensor contraction expressions into efficient, parallel Fortran code. A tensor contraction can be viewed as a sum of product of multi-dimensional arrays (often three to six dimensional arrays). A pair of optimizations in the TCE, the fusion and tiling optimizations, have proven successful for minimizing disk-to-memory traffic for dense tensor computations. These take advantage of the semantics of the tensor contraction expressions. Measurements of the code generated by TCE showed that in addition to optimizing tile sizes of the fused loop structure, it is necessary to optimize the layout and to select appropriate library calls for computing the inner-most loop nests. These library calls can be used to replace the generated code for the contraction. Unfortunately, existing linear algebra libraries implement optimized code for operations on two-dimensonal arrays (i.e., matrices).

A tensor contraction can be implemented by index permutations of its arguments to view each tensor as a two-dimensional matrix followed by a DGEMM matrix multiplication library call. There are typically several possible combinations of index permutations and DGEMM calls with different costs. We [LG+ 12] have previously developed a layout optimization algorithm. We measured the performance of the code structures resulting from our layout optimization algorithm on a variety of architectures.

10. Analytical Bounds to Improve Tile Size Selection

Modern computer systems utilize multi-level memory hierarchies in which the latency of data access from higher levels are orders of magnitude higher than the time required to perform arithmetic operations. Loop Tiling is a classical technique to enhance data reuse in memory hierarchy levels close to the processor. Recent advances have made it possible to automatically generate parametrically tiled code, even for imperfectly nested loops. It is well known that the choice of tile sizes has a significant effect on performance, but the effective selection of optimized tile sizes remains an open problem that has become ever more challenging as processor memory hierarchies increase in complexity and depth.

Past work has pursued two main types of approaches for tile size selection, analytical and empirical. In analytical approaches, a compiler selects tile sizes based on static analysis of loop nests and known characteristics of the memory hierarchy. Although several analytical techniques for tile size selection have been proposed in the literature, none has been demonstrated to be sufficiently effective for use in practice. As a result, the gap between the performance delivered by the best known tile sizes and those selected by an analytical approach has continued to widen, thereby diminishing the utility of past analytical approaches.

Empirical approaches to tile size optimization treat the loop nest as a black box, and perform empirical auto-tuning for a given architecture by actually executing the tiled code for a range of different tile sizes. The highly successful ATLAS (Automatically Tuned Linear Algebra Software) system uses empirical tuning at library installation time to find the best tile sizes for different problem sizes on the target machine. One of the most challenging issues in empirical approaches for tile size selection is the enormous search space that must be explored when tiling multiple loops. Though many empirical tuning systems attempt to reduce the search space by only examining square tiles, our results in [SS+ 12] reaffirm the importance of including non-square tiles in the search space.

In [SS+ 12], we introduce a novel approach using analytical bounds to limit the search space with empirical tuning for square and non-square tiling. The approach developed in [SS+ 12] to pruning the search space is complementary to, and can be combined with, existing empirical search strategies; e.g., the analytical bounds can be integrated with existing auto-tuning frameworks such as ATLAS. Experimental results show that our approach can reduce the search space by up to four orders of magnitude.

Our approach employs a pair of analytical models to prune the search space: a conservative model that underestimates the number of iterations in an optimal tile (DL), and an optimistic model that overestimates the number of iterations in an optimal tile (ML). DL (Distinct Lines), a conservative model from past work, models the required cache capacity for a tile as its total data footprint. Under this model, any tiles with a data footprint larger than the cache size are discarded, since they may incur capacity misses during execution. However, this is a pessimistic assumption for many applications, especially applications with streaming data accesses. We therefore introduced an optimistic analytical model, ML (Minimum working set Lines), that assumes an ideal intra-tile cache block replacement. Because DL and ML respectively provide lower and upper bounds for tile sizes, we can use them to bound tile size search space for empirical tuning. Our experiments show that this bounded search space still contains optimal tile sizes, despite reductions of up to four orders of magnitude in the size of the search space.

11. Domain-Specific Compilation of Stencil Codes

Stencils represent an important computational pattern used in scientific applications in a variety of domains including computational electromagnetics, solution of PDEs using finite difference or finite volume discretizations, and image processing for CT and MRI imaging. There is increasing interest in developing domain-specific frameworks for high-performance scientific computing due to the diversity of current/emerging parallel architectures. Using the Stencil Domain Specific Language (SDSL) we recently proposed, our recent work [HV+ 13] describes a set of compiler transformations that are needed to generate efficient code for (i) GPUs such as NVIDIA GTX580 and AMD A8-3850, and (ii) multicore processors with short-vector SIMD ISAs such as SSE, AVX, VSX, LRBNi etc., for stencil computations. Vector operations, with ISAs like SSE or AVX, require the loading of physically contiguous data elements from memory into vector registers and the execution of identical and independent operations on the components of vector registers. As stencil computations involve arithmetic operations on physically contiguous data elements, e.g., c0*(A[i-1]+A[i]+A[i+1]), they pose challenges to efficient implementation on these architectures. It requires the use of redundant and unaligned loads of data elements from memory into different slots. We had previously addressed this issue through a dimension-lifting-transpose (DLT) data layout transformation. However, only sequential, L1-resident execution was addressed. In [HV+ 13], we describe an integrated approach to perform tiling in conjunction with DLT transformation to generate efficient parallel code for stencil computations over large data sets on shared memory multiprocessors. We compare performance with code generated by the Pochoir stencil compiler and Pluto for several benchmarks on multiple target multicore processors, demonstrating strong performance benefits for 1D and 2D stencils.

12. Split-tiling for GPUs

Tiling is a key technique to enhance data reuse. For computations structured as one sequential outer time loop enclosing a set of parallel inner loops, tiling only the parallel inner loops may not enable enough data reuse in the cache. Tiling the inner loops along with the outer time loop enhances data locality but may require other transformations like loop skewing that inhibit inter-tile parallelism. One approach to tiling that enhances data locality without inhibiting inter-tile parallelism is split tiling, where tiles are subdivided into a sequence of trapezoidal computation steps. In [CG+ 13], we developed an approach to generate split tiled code for GPUs in the PPCG polyhedral code generator. We proposed

a generic algorithm to calculate index-set splitting that enables us to perform tiling for locality and synchronization avoidance, while simultaneously maintaining parallelism, without the need for skewing or redundant computations. Our algorithm performs split tiling for an arbitrary number of dimensions and without the need to construct any large integer linear program. The method and its implementation are evaluated on standard stencil kernels and compared with a state-of-the-art polyhedral compiler and with a domain-specific stencil compiler, both targeting CUDA GPUs. The newly developed algorithm is being incorporated into the publicly available PPCG tool flow.

13. Automatic Parallelization of Irregular Computations for Distributed Memory Systems

Parallelization and locality optimization of affine loop nests has been successfully addressed for shared-memory multiprocessors and GPUs. However, most large-scale simulation applications must be executed in a distributed-memory environment, and often use irregular/sparse computations where the control-flow and array-access patterns are data-dependent. In [RE+ 12], we developed compiler support for effective parallelization of such applications for a distributed-memory environment, using a combination of static and run-time analysis. We discussed algorithms to generate executor code to execute the original computation on multiple processes, along with inspector code that analyzes the data-dependent control-flow and arrayaccess patterns at run time and in parallel. The effectiveness of the framework was demonstrated on several benchmarks and a production climate modeling application.

14. Storage Optimization for Embedded Processors

Most modern digital signal processors (DSPs) provide multiple address registers and a dedicated address generation unit (AGU) which performs address generation in parallel to instruction execution. There is no address computation overhead if the next address is within the auto-modify range. A careful placement of variables in memory (called "offset assignment" or OA) is utilized to decrease the number of address arithmetic instructions and thus to generate compact and efficient code. The simple offset assignment (SOA) problem concerns the layout of variables for machines with one address register and the general offset assignment (GOA) deals with multiple address registers. Both these problems assume that each variable needs to be allocated for the entire duration of a program. Both SOA and GOA are NP-complete.

In [SR 11], we present effective heuristics for the simple and the general offset assignment problem with variable coalescing. Coalescing allows two or more variables to share the same memory location provided that their live ranges do not overlap. Based on the live ranges of all the variables, an interference graph is constructed where an edge (u, v) in the graph indicates that variables u and v interfere and thus they cannot be mapped into the same memory location. Variable coalescing, in which two or more non-interfering variables are mapped into the same memory location, improves the results by decreasing the number of address arithmetic instructions needed as well as the memory requirement for storing the variables. Results on several benchmarks show the significant improvement of our proposed heuristics compared to other heuristics in the literature.

In [WR 10], we consider the problem of SOA based on specialized DSPs with Address Generation Units (AGUs). A number of SOA algorithms had been proposed to solve the SOA problem in the past years. In this paper, a new heuristic has been proposed for SOA which dynamically selects the edge by re-sorting the edge array iteratively. A new technique for edge selection approach has been proposed to reduce the OA cost in advance. Experimental results on several benchmarks shows that our approach is not only better than the previous works but also can be applied onto the other OA algorithms to have a significant improvement.

In [SRa 12], we develop an integer linear programming solution for the address code generation problem for embedded processors, in particular for Digital Signal Processors (DSPs). In [SRb 12], we develop an approach for code size reduction for array-intensive applications on DSPs that reduces the size of the address code.

15. Task Scheduling and Memory Partitioning for Multi-Processor System-on-Chip

The growing trend in current complex embedded systems is to deploy multiprocessor system-on-chip (MPSoC). An MPSoC consists of multiple heterogeneous processing elements, a memory hierarchy, and input/output components which are linked together by an on-chip interconnect structure. Such architecture provides the flexibility to meet the performance requirements of multimedia applications while respecting the constraints on memory, cost, size,

time, and power. Many embedded systems employ software-managed memories known as scratch-pad memories (SPM). Scratchpad memories, unlike caches, are software-controlled and hence the execution time of applications on such systems can be accurately predicted and controlled. Scheduling the tasks of an embedded application on the processors as well as partitioning the available SPM budget among these processors are two critical issues in reducing the overall computation time as well as the communication overhead. Traditionally, the step of task scheduling is applied separately from the memory partitioning step. Such a decoupled approach may miss better quality schedules. In [SRc 12], we present an integrated approach to task scheduling and SPM partitioning to further reduce the execution time of embedded applications. Results on several real life benchmarks show the significant improvement of our technique compared to decoupled techniques as well as to an integer-linear programming approach.

References

[BH+ 10] M. Baskaran, A. Hartono, T. Henretty, S. Tavarageri, J. Ramanujam, and P. Sadayappan, "Parameterized Tiling Revisited," in *Proc. CGO 2010, The 8th International Symposium on Code Generation and Optimization,* pp. 200–209, Toronto, Canada, April 2010.

[BRS 10] M. Baskaran, J. Ramanujam, and P. Sadayappan, "Automatic C-to-CUDA Code Generation for Affine Programs," in *Proc. CC 2010 - International Conference on Compiler Construction,* Paphos, Cyprus (R. Gupta Ed.), Lecture Notes in Computer Science, Vol. 6011, pp. 244–263, Springer-Verlag, March 2010.

[HB+ 09] A. Hartono, M. Baskaran, C. Bastoul, A. Cohen, S. Krishnamoorthy, B. Norris, J. Ramanujam, and P. Sadayappan, "Parametric multi-level tiling of imperfectly nested loops," ACM International Conference on Supercomputing (ICS), New York, NY, June 2009.

[HB+ 10] A. Hartono, M. Baskaran, J. Ramanujam, and P. Sadayappan, "DynTile: Parametric Tiled Loop Generation for Parallel Execution on Multicore Processors," in *Proc. 24th International Parallel and Distributed Processing Symposium (IPDPS 2010),* Atlanta, GA, April 2010.

[TH+ 10] S. Tavarageri, A. Hartono, M. Baskaran, L.-N. Pouchet, J. Ramanujam, and P. Sadayappan, "Parametric Tiling of Affine Loop Nests," in *Proc. 15th Workshop on Compilers for Parallel Computers (CPC 2010),* Vienna, Austria, July 2010.

[HS+ 11] T. Henretty, K. Stock, L.-N. Pouchet, F. Franchetti, J. Ramanujam, and P. Sadayappan, "Data Layout Transformation for Stencil Computations on Short SIMD Architectures," in *Proc. CC 2011 - International Conference on Compiler Construction*, (J. Knoop Ed.), Lecture Notes in Computer Science, Vol. 6601, pp. 223–242, Springer-Verlag, 2011.

[PB+ 10] L.-N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam and P. Sadayappan, "Combined Iterative and Model-driven Optimization in an Automatic Parallelization Framework," in *Proc. the ACM/IEEE Conference on High Performance Computing SC10*, New Orleans, LA, November 2010.

[PB+ 11] L.-N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, P. Sadayappan and N. Vasilache, "Loop Transformations: Convexity, Pruning and Optimization," in *Proc. Symposium on Principles of Programming Languages (POPL)*, pp. 549–561, Austin, TX, January 2011.

[SR 11] H. Salamy and J. Ramanujam, "Storage Optimization through Offset Assignment with Variable Coalescing," *ACM Transactions on Embedded Computing Systems,* 2011.

[WR 10] T. Wang and J. Ramanujam, "A Dynamic Heuristic Algorithm for Offset Assignment," in *Proc. International Computer Symposium,* Tainan, Taiwan, December 2010.

[LG+ 12] Qingda Lu, Xiaoyang Gao, Sriram Krishnamoorthy, Gerald Baumgartner, J. Ramanujam, P. Sadayappan, "Empirical Performance-Model Driven Data Layout Optimization and Library Call Selection," *Journal of Parallel and Distributed Computing*, 72(3):338–352, March 2012.

[SRa 12] H. Salamy and J. Ramanujam, "An ILP Solution to Address Code Generation for Embedded Applications on Digital Signal Processors," *ACM Transactions on Design Automation of Electronic Systems*, vol. 17, no. 3, June 2012.

[SRb 12] H. Salamy and J. Ramanujam, "Code Size Reduction for Array Intensive Applications on Digital Signal Processors," *Journal of Circuits, Systems, and Computers*, vol. 21, no. 3, pp. 1–22, 2012.

[SRc 12] H. Salamy and J. Ramanujam, "An Effective Solution to Task Scheduling and Memory Partitioning for Multi-Processor System-on-Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 717–725, May 2012.

[SS+ 12] J. Shirako, K. Sharma, N. Fauzia, L.-. Noel Pouchet, J. Ramanujam, P. Sadayappan and V. Sarkar "Analytical Bounds for Optimal Tile Size Selection," in *Proc. CC 2012 - International Conference on Compiler Construction*, (M. O'Boyle Ed.), Lecture Notes in Computer Science, Vol. 7210, pp. 101–121, Springer-Verlag, 2012.

[TP+ 11] S. Tavarageri, L.-N. Pouchet, J. Ramanujam, A. Rountev, and P. Sadayappan, "Dynamic selection of tile sizes," in *Proc. 18th Annual International Conference on High Performance Computing*, Bangalore, India, December 2011.

[TR+ 13] S. Tavarageri, J. Ramanujam, and P. Sadayappan, Adaptive Parallel Tiled Code Generation and Accelerated Auto-tuning, International Journal of High Performance Computing, 2013.

[HV+ 13] T. Henretty, R. Veras, F. Franchetti, L.N. Pouchet, J. Ramanujam and P. Sadayappan, A Stencil Compiler for Short-Vector SIMD Architectures, in Proc. 27th ACM International Conference on Supercomputing, Eugene, OR, June 2013.

[CG+ 13] A. Cohen, T. Grosser, P. Kelly, J. Ramanujam, P. Sadayappan, and S. Verdoolaege, Tiling for GPUs: Automatic Parallelization Using Trapezoidal Tiles to Reconcile Parallelism and Locality, Avoiding Divergence and Load Imbalance, in Proc. 6th Workshop on General Purpose Processing Using GPUs (GPGPU-6), held with ASPLOS 13, March 2013.

[RE+ 12] M. Ravishankar, J. Eisenlohr, L.-N. Pouchet, J. Ramanujam, A. Rountev, and P. Sadayappan, Automatic Parallelization of a Class of Extended Affine Loop Nests for Distributed Memory Systems, in Proc. ACM/IEEE Conference on High Performance Computing SC12, Salt Lake City, UT, November 2012.